Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Luca Alves da Silva

Graph-Based Clustering In Deep Feature Space for Shape Matching

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor : Prof. Waldemar Celes Filho
Co-advisor: Prof. Paulo Ivson Netto Santos

Rio de Janeiro
April 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Luca Alves da Silva

Graph-Based Clustering In Deep Feature Space for Shape Matching

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Waldemar Celes Filho
Advisor
Departamento de Informática – PUC-Rio

Prof. Paulo Ivson Netto Santos
Co-advisor
Departamento de Informática – PUC-Rio

Prof. Alberto Barbosa Raposo
Departamento de Informática – PUC-Rio

Prof. Anselmo Cardoso de Paiva
Departamento de Informática – UFMA

Dr. Lucas Caracas de Figueiredo
Tecgraf – PUC-Rio

Rio de Janeiro, April 19th, 2024

Daniel Luca Alves da Silva

Daniel Luca da Silva received his Bachelor's degree in Computer Engineering from Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) in 2020.

Bibliographic data

Alves da Silva, Daniel Luca

Graph-Based Clustering In Deep Feature Space for Shape Matching / Daniel Luca Alves da Silva; advisor: Waldemar Celes Filho; co-advisor: Paulo Ivson Netto Santos. – 2024.

55 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. Informática – Teses. 2. Modelos CAD 3D. 3. Nuvens de ponto. 4. Correspondência de formas. 5. Aprendizado profundo. 6. Clusterização. I. Celes Filho, Waldemar. II. Ivson, Paulo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my family, my friends, and all the people
I have loved and will love.

Acknowledgments

First, I would like to express my gratitude to my advisor Waldemar Celes for his trust, and patient mentorship throughout the course of this research.

I would like to thank my co-advisor Paulo Ivson for his shared knowledge, and sharp understanding of our problem domain.

I would like to thank my parents, Amilton and Maria, as well as my siblings, Daiana, Aaron, and Laura, for the love and support they always provided me.

I would like to thank my friends Bianca, Leonardo, and Eduardo, for their kindness and encouragement when I needed the most. I am also grateful to Vinícius, my friend and fellow student who shared many experiences alongside me these past couple of years.

Thanks to PUC-Rio and Tecgraf, for the financial assistance and support.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Abstract

Alves da Silva, Daniel Luca; Celes Filho, Waldemar (Advisor); Ivson, Paulo (Co-Advisor). **Graph-Based Clustering In Deep Feature Space for Shape Matching**. Rio de Janeiro, 2024. 55p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Engineering projects rely on complex 3D CAD models throughout their life cycle. These 3D models comprise millions of geometries that impose storage, transmission, and rendering challenges. Previous works have successfully employed shape-matching techniques based on deep learning to reduce the memory required by these 3D models. This work proposes a graph-based algorithm that improves unsupervised clustering in deep feature space. This approach dramatically refines shape-matching accuracy and results in even lower memory requirements for the 3D models. In a labeled dataset, our method achieves a 95% model reduction, outperforming previous unsupervised techniques that achieved 87% and almost reaching the 97% reduction from a fully supervised approach. In an unlabeled dataset, our method achieves an average model reduction of 87% versus an average reduction of 77% from previous unsupervised techniques.

Keywords

3D CAD models; Point cloud; Shape matching; Deep learning; Clustering.

Resumo

Alves da Silva, Daniel Luca; Celes Filho, Waldemar; Ivson, Paulo. **Clusterização Baseada em Grafo em Espaço de Características Profundo para Correspondência de Formas**. Rio de Janeiro, 2024. 55p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Projetos de engenharia dependem de modelos CAD 3D complexos durante todo o seu ciclo de vida. Esses modelos 3D contêm milhões de geometrias que impõem desafios de armazenamento, transmissão e renderização. Trabalhos anteriores empregaram com sucesso técnicas de correspondência de formas baseadas em aprendizado profundo para reduzir a memória exigida por esses modelos 3D. Este trabalho propõe um algoritmo baseado em grafos que melhora o agrupamento não supervisionado em espaços profundos de características. Essa abordagem refina drasticamente a precisão da correspondência de formas e resulta em requisitos de memória ainda mais baixos para os modelos 3D. Em um conjunto de dados rotulado, nosso método atinge uma redução de 95% do modelo, superando as técnicas não supervisionadas anteriores que alcançaram 87% e quase atingindo a redução de 97% de uma abordagem totalmente supervisionada. Em um conjunto de dados não rotulado, nosso método atinge uma redução média do modelo de 87% contra uma redução média de 77% das técnicas não supervisionadas anteriores.

Palavras-chave

Modelos CAD 3D; Nuvens de ponto; Correspondência de formas; Aprendizado profundo; Clusterização.

Table of contents

List of figures

List of tables

List of algorithms

List of Abreviations

BIM – Building Information Modeling

CAD – Computer Aided Design

CNN – Convolutional Neural Network

FPS – Farthest Point Sampling

MLP – Multi-Layer Perceptron

MWDDS – Minimum Weighted Directed Dominating Set

MWSC – Minimum Weighted Set Cover

And the only word there spoken was the
whispered word, "Lenore?"
This I whispered, and an echo murmured
back the word, "Lenore!" —
Merely this and nothing more.

Edgar Allan Poe, *The Raven.*

1
Introduction

Computer-Aided Design models have a pivotal role in modern industrial practice. Their usage is present throughout various phases of engineering endeavors, starting from design, building, and subsequent operation. The prevalence of Building Information Modeling (6), in conjunction with the growing adoption of Digital Twins for smart manufacturing (7), resulted in an increasing need for more intricate and comprehensive 3D CAD models, as illustrated by Fig. 1.1. Consequently, such models' storage, transmission, and rendering have become a challenge even to modern hardware capabilities.

These models have many regular geometries, meaning many geometric shapes inside them could refer to one triangle mesh and many transformation matrices. That is, many meshes are instances of the same base mesh. However, instancing information is usually absent in such models. As a result, instance detection is a common and vital preprocessing stage before they are delivered to final users. The redundancy detection also enables rendering performance improvements using hardware-accelerated instance rendering (8).

Ivson and Celes (8) exploited geometric regularity to reduce the sizes of models. Their technique was based on least squares optimization. Their work accomplished impressive model reductions, but their method heavily relies on mesh topology, as the error evaluation depends on the number of vertices and

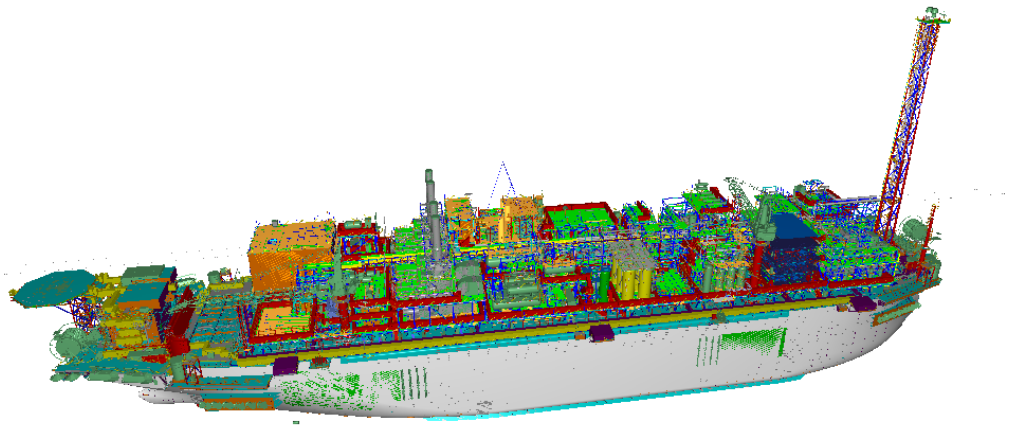


Figure 1.1: Example of CAD model with close to 9 million geometries

their ordering. Figueiredo et al. (9) proposed a deep learning-based framework for shape matching that has overcome the dependency on mesh topology but relied on large annotated data sets. Their work was based on deep learning models that processed uniform point clouds sampled on the surfaces of the 3D CAD geometries. Most recently, Figueiredo et al. (5) proposed an unsupervised deep learning-based framework that has overcome the need for large annotated data sets through unsupervised feature learning and unsupervised clustering algorithms but falls short in model reduction when compared to the supervised counterpart.

Our work builds on top of the existing unsupervised shape-matching framework presented by Figueiredo et al. (5) to further exploit the similarity of geometries close in the feature space. We achieve significantly better results by changing the clustering step. We construct a graph that captures relationships between meshes and derive clusters based on this representation.

Once we can determine which meshes are instances of other meshes, the goal of reducing the model size can be achieved by finding the smallest subset of meshes required to represent the remaining meshes of the model. This is a problem of Minimum Weighted Set Cover (MWSC), a well-known NP-Hard problem (10). An alternative way to model our problem is as a Minimum Weighted Directed Dominating Set (MWDDS). We design an algorithm to represent our data in a directed graph such that every mesh corresponds to a node with an associated weight, and an arc connects a node to every one of its instances. We then exploit the connections in the graph to extend our knowledge of instances and find an approximate solution to the MWDDS while evading to compute all arcs.

Our proposed graph-based algorithm improves unsupervised clustering, resulting in greater shape-matching accuracy and even lower 3D model sizes. We verify that our method achieves an average model reduction of 87% in an unlabeled dataset versus an average reduction of 77% from previous unsupervised techniques. In a labeled dataset, our method achieves a 95% model reduction, outperforming previous unsupervised techniques that acquired 87% and almost getting as far as the 97% reduction from a fully supervised approach, effectively bridging the gap between both.

The structure of this document is as follows: Chapter 2 introduces the core concepts of this work; Chapter 3 discusses previous works on shape-matching techniques and geometry registration; In Chapter 4, we present our shape-matching algorithm; In Chapter 5 we detail our experiments, the model reductions we achieved and highlight critical aspects of our work in comparison to previous ones; In Chapter 6, we conclude and propose future work.

2
Theoretical Background

This chapter showcases concepts and building block techniques for our work. We introduce basic notions of deep neural networks and autoencoders. Afterward, we detail two neural network architectures tailored specifically for processing point cloud data. After that, we briefly discuss the idea of clustering and mention two clustering algorithms. Then, we formulate the shape-matching problem as a minimum dominating set problem and justify our algorithm design choices.

2.1
Deep Neural Networks

Neural networks are mathematical models that draw their name and inspiration from neuroscience (1). The building blocks of neural networks are interconnected neurons that apply individual transformations to their input. Deep neural network models are referred to as deep models because they contain several intermediate layers between input and output, called hidden layers. These layers can learn progressively low-level to high-level patterns of the data.

A fundamental example of a deep learning model is the multilayer perceptron (MLP). An MLP is a model of stacked layers of neurons with links connecting every neuron in each layer to every neuron in the subsequent layer. MLPs are a type of feedforward network because there is no feedback link in its layers. For example, MLP architectures have been used to tackle problems in the healthcare domain related to breast cancer (11), Parkinson's disease, diabetes, coronary heart disease, and orthopedic patients classification (12). It has been proven that MLPs can approximate any function (13), however, for practical reasons, this architecture may be inadequate for some domains where specific architectures have been demonstrated more successful.

One such example is convolutional neural networks (14). This type of network is based on the mathematical operation called convolution, which processes input in a spatially sensitive manner. CNNs have been successfully applied to grid-like data like images (15) or time-series (16). One of the reasons for the success of CNNs is their ability to learn hierarchical patterns (17).

In Fig. 2.1, we can see an illustration of a deep learning model with the patterns that activate its hidden layers. Typically, the last layers of a CNN

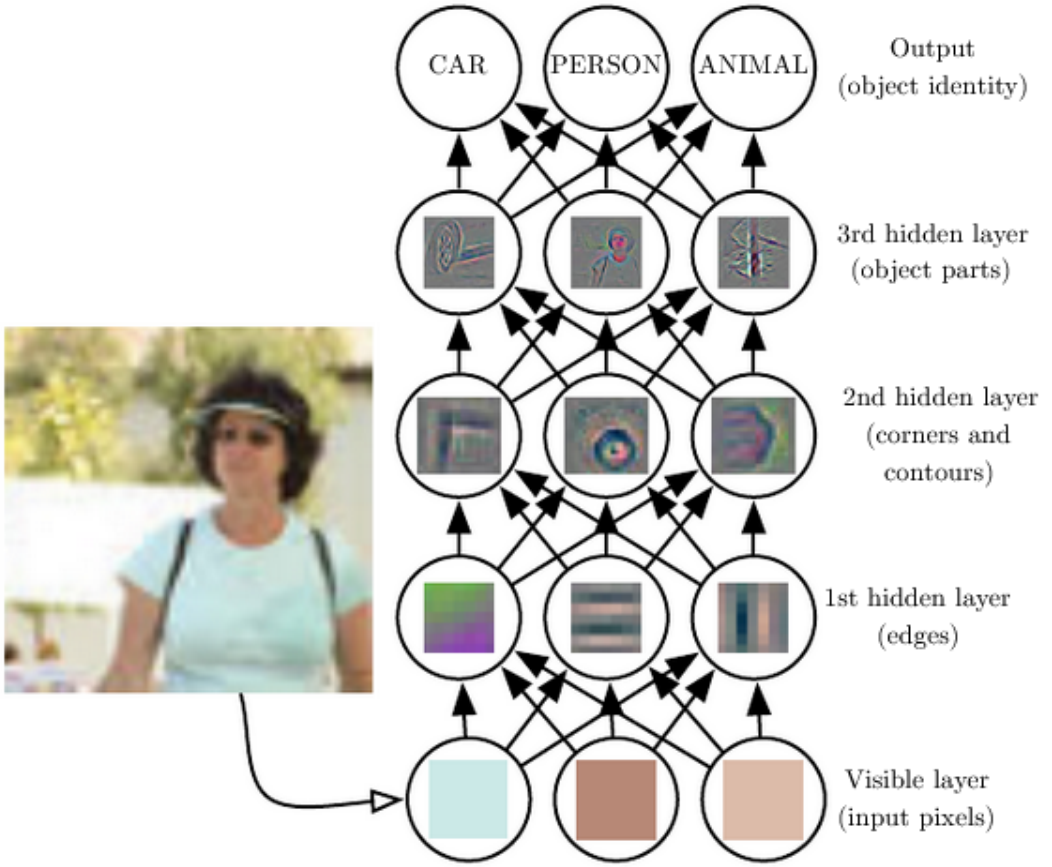


Figure 2.1: Illustration of a deep learning model. Image extracted from (1).

are flattened and fed to fully connected layers that are exactly like an MLP network.

2.2 Autoencoders

Autoencoders are a type of neural network that aims to resolve the problem of extracting meaningful information from real-world data without annotation. Autoencoders are structured as a combination of two parts: the encoder and the decoder. Both parts are trained in conjunction to respectively learn to transform data into a representation space and reconstruct the input data from this representation. The output layer of the encoder is the input to the decoder and is denoted as the bottleneck. The bottleneck is the layer that outputs a "code" for the input. This code can also be referred to as the feature vector.

An autoencoder is undercomplete when the bottleneck has a lower dimension than the input data. A reduced dimension helps to achieve the autoencoder's goal as it forces the encoder to select only the most critical

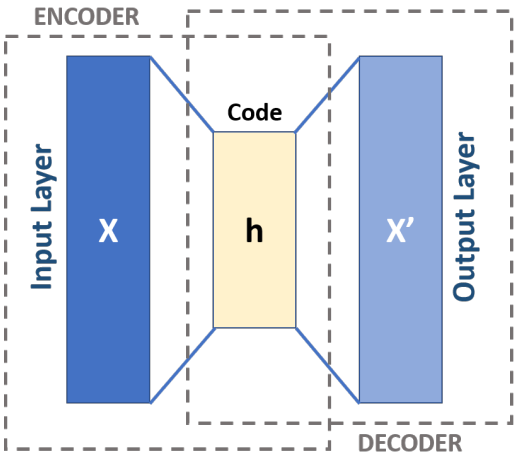


Figure 2.2: Schema of an autoencoder. Michela Massi, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

features of the input, as the decoder needs these features in order to reconstruct the original data. As such, an autoencoder can be used to compress data.

If the bottleneck has a dimension equal to or higher than the input data, however, an autoencoder is said to be overcomplete. An overcomplete autoencoder can be useful for learning a wider range of features, but it requires the usage of regularization techniques to prevent the network from learning to encode the input with the identity function because in such case the code would be identical to the input and consequently useless.

Autoencoders can also be helpful for various tasks like classification (18), regression (19), and clustering (20). They can also be used for transfer learning if they are trained on large general datasets and then fine-tuned for tasks in domain-specific smaller ones.

They are often used as building blocks for other deep-learning models. Autoencoders are the foundation for variational autoencoders, a technique that can be used for data generation. Pre-trained autoencoders can also be used to improve the stability of training generative adversarial networks .

2.3 PointNet and PointNet++

PointNet (21) was the first deep learning architecture to process raw point cloud data. At the time, most researchers would transform point clouds into regular 3D voxel grids or collections of images and then process them with a deep network architecture. However, quantization artifacts arise from regular-grid data representation, and such artifacts can conceal natural invariances of the data. PointNet takes point clouds as input and outputs either class labels for the entire point cloud or segments the input with per-point labels.

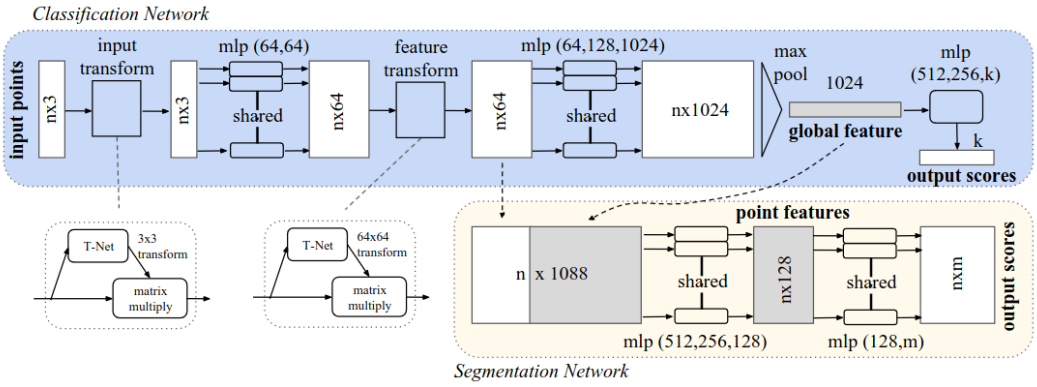


Figure 2.3: The PointNet architecture. The classification network operates on a set of points, applying transformations to them and aggregating their features. This process generates scores for various categories. The segmentation network, an extension of the classification framework, combines global and local features to provide scores for individual points. "mlp" refers to a multilayer perceptron, with the sizes of its layers specified within brackets. Image adapted from (2).

In the initial layers of PointNet, each point is processed identically and independently in an MLP. Then, PointNet achieves permutation invariance by adding symmetric functions in the net computation. So, the network learns a set of optimization functions that pick meaningful points of the point cloud. The final fully connected layers of the network combine these learned values to output class labels. As a result, the network learns to select a sparse set of key points that roughly resembles the skeleton of objects. PointNet learns to probe the space so that noise or small corruptions in the input set are not expected to change the network's output.

As the basic idea of PointNet is to learn a spatial encoding of each point and then aggregate all individual point features to a global point cloud signature, it does not capture local structures yielded by the metric space in which points lie. This results in a limited ability to recognize fine-grained patterns and to generalize to complex scenes. To address this issue, PointNet++ has been proposed (3). It has a hierarchical neural network that applies PointNet recursively on partitions of the input point set.

Inspired by CNN architectures, PointNet++ progressively captures features along a multi-resolution hierarchy at increasingly larger scales. These local features are grouped into larger units to produce higher-level features. PointNet++ uses PointNet as a local feature learner that shares weights when processing features of the same level, like convolutional networks.

PointNet++ applies three layers at each level to process the input data progressively: the sampling, grouping, and PointNet layers. The sampling layer selects a subset of points from the input set as centroids of local regions

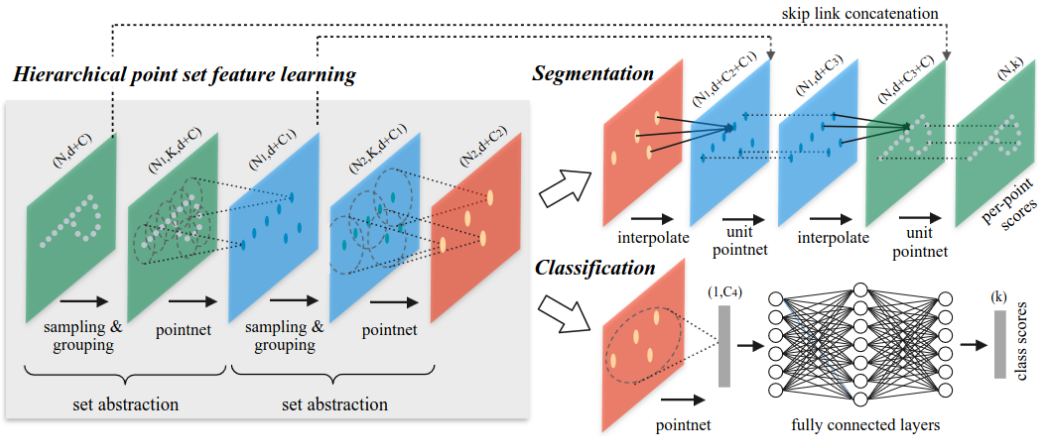


Figure 2.4: PointNet++ architecture. The network progressively abstracts the points from a local to a wider receptive field through sampling, grouping, and PointNet layers. Image adapted from (3).

with farthest point sampling (FPS) to guarantee its appropriate coverage. The grouping layer then constructs local region sets by finding all points within a radius of the query point. The grouping layer will encounter a distinct number of neighbors for each centroid. This way, a fixed region scale is guaranteed for the local neighborhood. The PointNet layer uses a mini-PointNet to encode the selected points into feature vectors and is flexible enough to output fixed-length vectors besides the varying number of inputs.

Unlike CNNs, PointNet++ applies transformations in a data-dependent manner since the local regions depend on the selection of centroids from input data. PointNet++ successfully extracts multi-scale patterns by concatenating summarized features from each sub-region to a global feature vector obtained by processing the whole set of input points with a single PointNet, using a three-level hierarchical network with three fully connected layers.

2.4 Clustering

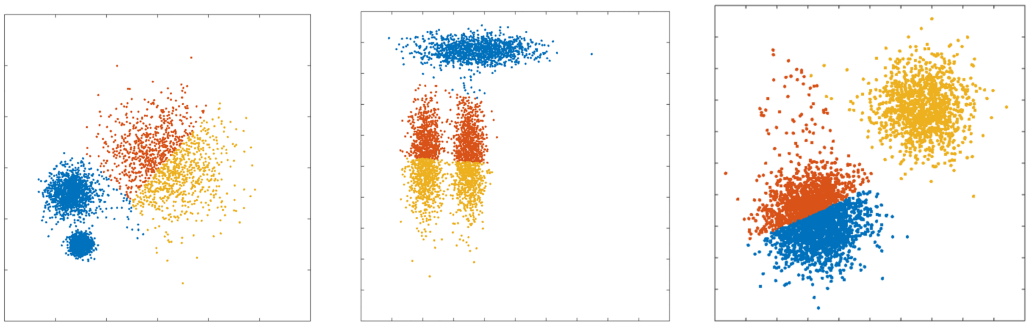
Clustering algorithms are used to group unlabeled data into meaningful subsets. Their core concern is to extract the underlying structure of the data given as input. There are many approaches to this problem (22) like algorithms based on partitions, hierarchy, density, or graphs, to name a few. Generally, the goal of a clustering algorithm is to minimize clusters' internal similarity and maximize inter-cluster dissimilarity while abiding by a precise and practical measurement of similarity.

2.4.1 K-Means‖

K-Means is a popular centroid-based clustering algorithm. It takes the number of clusters to be generated and a set of points as input. The first step of K-Means is randomly initializing values for centroids. In the second step, K-Means assigns each point to the cluster represented by the nearest centroid. During the third step, the centroids of each cluster are updated with the average of all elements inside the cluster. Then, the second and third steps are repeated until there is no change in the composition of clusters anymore.

The K-Means‖ (23) algorithm is a variant of K-Means. It also takes the number of generated clusters as a parameter. The main difference between K-Means‖ and K-Means lies in the initialization of the centroids. While the original algorithm selects random centroids, the K-Means‖ selects the initial centroids with a probability proportional to the squared distance to the nearest cluster centroid, a strategy first proposed by K-Means++ (24). The novelty of K-Means‖ over K-Means++ is a parallel version of the algorithm that improves scalability.

All K-Means variations share the limitation of being sensitive to the number of clusters given as input, not being well suited to non-spherical data, sensitivity to outliers, potential local minima, or density variation amongst clusters.



(a) Spherical, synthetic Gaussian data, with unequal cluster radii and density

(b) Synthetic elliptical Gaussian data.

(c) Spherical, synthetic Gaussian data

Figure 2.5: Examples where K-Means fails to produce good clusters. Images extracted from (4).

2.4.2 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise, or HDBSCAN (25), is a clustering algorithm based on density and hierarchy. This algorithm extends DBSCAN (26), a density-based algorithm. The core idea is that regions with many concentrated elements should belong to the same cluster. HDBSCAN can be conceptually understood as a search for clusters that persist over all density values and consequently dismisses the need for the density parameter required by DBSCAN.

The algorithm relies on the concept of a *core distance* to define a metric called *mutual reachability distance*. The core distance κ depends on a parameter m_{pts}, as it is the distance d from a point x to its m_{pts}-th nearest neighbor. The distance d can be the Euclidean distance. The mutual reachability distance d_{mreach} is given below for $i \neq j$.

$$d_{mreach}(x_i, x_j) = max(\kappa(x_i), \kappa(x_j), d(x_i, x_j)) \tag{2-1}$$

This metric is used to construct a *mutual reachability graph* denoted G_{ms} associating each point x from the input to a node n of the graph, and the weight of edge (n_i, n_j) to be the distance $d_{mreach}(x_i, x_j)$. The algorithm builds the minimum spanning tree (MST) of G_{ms} and considers this tree the first cluster.

From this point onward, the algorithm estimates a persistence value λ for clusters as the inverse of its edges' maximum weight ϵ. The algorithm proceeds to remove the edge of the greatest weight and evaluates the remaining connected components for each removal. The evaluation of each connected component is based on an integer parameter m, which defines the minimum number of nodes necessary to constitute a cluster. If only one connected component has more than m nodes, this connected component is considered to be the same cluster as the previous step. The components that have less than m nodes are considered noise. If more than one component has sufficient elements, both are considered two novel clusters.

As the removal of edges progresses, the value for ϵ decreases, the value for λ increases as a consequence, and the tree condenses. Finally, the stability σ of a cluster is defined in terms of the sum of the λ values of its nodes. Below, λ_{max} is the value for λ where x was removed from C_i, and λ_{min} is the value of λ where C_i appeared.

$$\sigma(C_i) = \sum_{x \in C_i} (\lambda_{max}(x) - \lambda_{min}) \tag{2-2}$$

Finally, the algorithm outputs the set of clusters as the non-overlapping

clusters that maximize the sum of stability. These clusters may be produced at any level of density. Usually, the parameter m is simplified to take the same value as m_{pts}.

2.5 Minimum Dominating Set Problem applied to Shape Matching

A digraph is denoted by $G = (V, A)$ where V is the set of nodes and A is the set of arcs. Each arc is an ordered pair (u, v) representing an edge that starts in u and ends in v. A triangle mesh T_v is a successfully detected instance of a mesh T_u if there is a matrix M, an error function E, and a threshold α such that $E(M*T_u, T_v) < \alpha$. Thus, a CAD model's geometries and instance relations can be represented in a digraph structure where each geometry corresponds to a node, and every arc represents a successful registration of the starting node to the ending node. Therefore, an arc (u, v) means that there is a matrix M_{uv} such that $E(M_{uv} * T_U, T_V) < \alpha$.

In a directed graph, a node u dominates a node v if the arc (u, v) exists. A set of nodes B is a dominating set of G if for every $v \in V \setminus B$ there is another node $u \in B$ such that the arc $(u, v) \in A$. The representation of the geometries of a model and its instance relations in a graph allows the reduction of the model size because we only need to represent the meshes associated with the nodes of the dominating set of the graph. The geometry of every dominated node can be represented by the base mesh associated with the node that dominates it and a known transformation matrix.

The smallest possible size of a model can be attained by finding the set with smallest size that can represent all geometries. In order for us to find such set we need the graph that encodes the geometric relations of the model, where the size of each node's geometry corresponds to the node's weight, and then solve the Minimum Weighted Directed Dominating Set (MWDDS) on this graph. The MWDDS can be reduced to the Minimum Weighted Set Cover (MWSC) where each node u results in a set S_U, such that $S_U \leftarrow \{u \cup w_0 \cup w_1 \cup ... \cup w_n\}$ and w_i are the nodes for which $(u, w_i) \in A$ and the weight for S_U is the weight for u. The MWSC is a well-known NP-Hard problem (10).

An alignment procedure can be used to compute an approximation matrix for every pair of geometries in a CAD model. If we encoded this information in a graph G^*, we could find the minimum number of base meshes necessary to represent every geometry in the model by solving the MWDDS on G^*. However, we find it impractical to compute all arcs on this graph. Computing an approximation matrix for a pair of geometries is costly and the

number of arcs in a complete graph is $O(n^2)$. In our approach, we proceed to compute a graph with a subset of arcs that connect potentially similar geometries.

There are greedy algorithms to the Minimum Weighted Set Cover (27, 28) as well as exact solutions (29). However, as stated by Albuquerque and Vidal (30), solutions to the MWSC generated from graphs with medium or high densities result in large sets that become unusually challenging for existing approaches. Hence, there is a need for more specific methods.

There exists work for the Minimum Directed Dominating Set on unweighted graphs (31), but the literature on approaches to the Minimum Weighted Directed Dominating Set (MWDDS) problem is not vast. Nakkala et al. (32) recently addressed this by presenting four approaches to this problem: two approaches based on swarm intelligence, one based on integer linear programming, and one matheuristic. There is a broader literature on how to solve the more specific undirected problem of Minimum Weighted Dominating Set (33, 34, 35, 30).

On the other hand, in this work we take advantage of the existing geometric relations between nodes to extend the number of edges in the initial graph and approximate the optimal solutions with heuristics. To do so, we design a three-stage clustering algorithm.

3
Previous Works

In this Chapter, we discuss previous works on shape-matching techniques applicable to the CAD domain. First, we explain a least squares algorithm and its limitations. Then, we mention deep-learning supervised works that can be used to reduce CAD models' sizes, and the instance registration procedure proposed by one of its works. Finally, we discuss an unsupervised shape-matching method and its adjustments to the instance procedure mentioned before.

Multiple works aim to detect similarities among 3D shapes. Among these, some are not well suited to the CAD domain because they do not take free-form scaling into account when detecting similarity (36, 37, 38, 39, 40, 41, 42, 43).

3.1
Least-Squares instance detection

Ivson and Celes (8) proposed a method for finding mesh instances based on least squares that can detect non-uniform scaling. However, their work has drawbacks regarding mesh topology, as their algorithm solely relies on the ordered vertices of the meshes as input. This results in error values sensitive to vertex ordering, even when the underlying mesh structure remains unchanged. Additionally, their method overlooks holes in the geometries.

They assume each triangle mesh to be an ordered set of vertices. For each pair of meshes with the same number of vertices, they compute an affine matrix M that minimizes the error given in (Equation 3-1). Below, q are the vertices of a possible instance and p are the vertices of a candidate base mesh.

$$E = \sum_{i} \|\mathbf{q_i} - \mathbf{M} * \mathbf{p_i}\|^2 \tag{3-1}$$

Their algorithm processes the meshes of the model sequentially and creates a set of base meshes. Each new processed mesh is compared to the known base meshes that have the same number of vertices and is considered as an instance if there is a transformation from the base meshes that comply with an error threshold given as input to their algorithm.

3.2
Supervised Deep Learning Approaches

There are already well-established deep-learning techniques that can reduce redundancy in 3D CAD models. The PointNet (2) and later PointNet++

(3) architectures propelled such advances by enabling successful segmentation and classification of 3D point clouds. Later work took advantage of these architectures by sampling point clouds on the surface of 3D CAD geometries, for which one example is the Supervised Primitive Fitting Network (SPFN) proposed by Li et al. (44). SPFN is built on top of PointNet++ to segment point cloud data into parametric surfaces such as planes, spheres, cones, and cylinders. It intends to fit parametric surfaces to point clouds scanned on real-world objects. Still, it can be used to reduce the size of CAD geometries represented by mesh triangles by subdividing them into parametric segments.

Figueiredo et al. (9) proposed a Deep Learning Framework powered by a PointNet++ classifier trained on a previously labeled data set. Each label has an associated base mesh that will replace the classified geometries in their work. Along with their classifier, they also contributed with an alignment procedure that computes the transformation matrix required to represent each instance mesh. Different meshes may have the same label in their data set yet not be aligned. Their framework is not limited to parametric surfaces but will not extend well to different models that lack label information.

3.2.1 Instance Registration

The approximation of a matrix that transforms a base mesh into a candidate instance requires the usage of a registration procedure and the definition of error functions. The registration procedure proposed by Figueiredo et al. (9) takes two normalized meshes as input: a base mesh B and a target mesh T. It performs an optimization that finds a 3x3 matrix M that fits the base mesh to the target mesh, such that the surface error function E_S between a dense point cloud P_{MB} sampled on the surface of $M * B$ to the target mesh T is minimized. The expression for E_S is given below, where N_Q is the number of points of the point cloud Q and $D_{\min}$ denotes the minimum distance from a point to the surface of T.

$$E_S(Q,T) = \frac{\sum_{q \in Q} D_{\min}(q,T)}{N_Q} \tag{3-2}$$

The optimization procedure starts with an alignment of both input meshes using Principal Component Analysis (PCA) and then proceeds to use the Adam (45) optimizer to update a matrix that transforms the base mesh into the target mesh. It consists of two nested loops that jointly drive the optimization. Their outer loop controls the optimization procedure, either interrupting the process when the desired surface error is achieved or increasing the density of the point clouds used to align the base mesh to the target mesh

when the inner loop finds a local minimum. It also guarantees that the point cloud sampled on the base mesh remains uniform. Meanwhile, the inner loop performs the actual optimization. It computes the gradients for the update of the matrix with the Chamfer pseudo-distance, a fast and differentiable function.

3.3 Unsupervised Shape Matching Method

To overcome the limitations of the Deep Learning supervised approach, Figueiredo et al. (5) have proposed a modified shape-matching framework that does not require annotated data sets. They rely on a PointNet++ autoencoder to do deep processing of point clouds and learn to extract meaningful features from point cloud data.

Their shape-matching method's first step is to extract feature vectors from the 3D CAD geometries by uniformly sampling point clouds on the surfaces of 3D geometries before feeding them to the deep neural network autoencoder. Taking the feature vectors as input, they perform an unsupervised clustering step and pick a base mesh for each cluster. Then, they use the alignment method first proposed by Figueiredo et al. (9) to register the meshes to their selected base mesh. They introduced a verification function with three error evaluations to guarantee that the resulting model remains faithful to the input, and unsuccessful registrations are discarded as instances and kept as individual meshes in the model. Their work successfully reduces models' sizes and even outperforms previous approaches on certain models, but stays akin to results presented by the supervised classification algorithm. Our work bridges that gap.

Their framework generalizes to any triangle mesh that may be present in the model, as they do not restrict the geometries to primitives like cones and cylinders. Their method also guarantees an upper bound on geometric errors, using tailored error functions to discard unsuccessful cluster assignments.

Their method is adequate for the 3D CAD domain as it takes into account affine transformations instead of just rigid-body transformations.

Instance identification through feature vectors may benefit from unsupervised clustering algorithms. Figueiredo et al. (5) has used K-Means‖ (23) and HDBSCAN (25) in this step. Other unsupervised clustering algorithms could be proposed, such as Birch (46) or Mean Shift (47). However, the disadvantage of relying on such algorithms is that they will assign each mesh to a cluster based on its distribution on the feature space alone and might be sensitive to outliers. In our work, on the other hand, we interpret the feature vectors as

nodes of a graph and extract subgraphs from it as a way to identify clusters.

3.3.1 Instance Registration Evaluation

Registration of $M * B$ into T will be rejected in the unsupervised framework if the final error is above certain thresholds. The functions evaluated are the symmetric surface error $SymE_S$ (Equation 3-3) and the maximum distance error $SymE_{max}$ (Equation 3-4), both evaluated with normalized meshes. In the expressions below P_{S1} and P_{S2} are dense point clouds sampled on the surfaces of P_1 and P_2, respectively. Additionally, the final area cannot differ above a certain threshold given in squared meters.

$$SymE_S(S_1, S_2) = \max(E_S(P_{S1}, S_2), E_S(P_{S2}, S_1)) \tag{3-3}$$

$$SymE_{max}(S_1, S_2) = \max(E_{max}(P_{S1}, S_2), E_{max}(P_{S2}, S_1)) \tag{3-4}$$

$$E_{max}(Q, T) = \max_{q \in Q}(D_{\min}(q, T)) \tag{3-5}$$

At the end of each registration procedure, T is considered an instance of B if the errors are below fine-tuned parameters reported by (5). The values should be $SymE_S(M * B, T) \leq 0.05$, $SymE_{max}(M * B, T) \leq 0.07$ and the absolute total area difference is below 0.01 squared meters.

4
Proposed Method

In this Chapter, we describe our proposed graph-based clustering algorithm. We detail the three stages of our method, provide a simplified visualization of how it proceeds, and present the adjustments we made to the former unsupervised shape-matching framework proposed by Figueiredo et al. (5). We elaborate on how we compose transformation matrices while traversing the graph and use such matrices to attempt to add arcs to the graph. We also detail the heuristics we used to pick base meshes in each connected component and explain the intuition behind them.

4.1
Clustering Algorithm

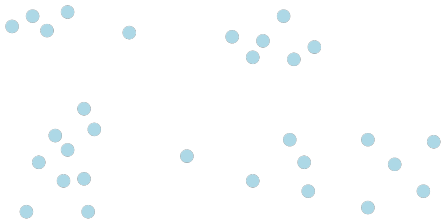
In this work, we propose a three-stage clustering algorithm that narrows down the number of comparisons between meshes, then groups similar meshes, and finally produces clusters. Each stage is described below:

1. Extract feature vectors from the geometries of a model and select the K nearest neighbors for each mesh in the feature space.

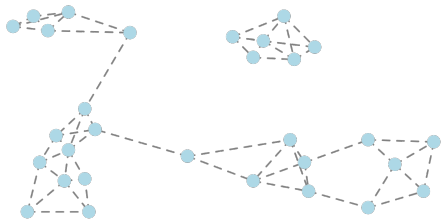
2. For each pair of neighbors T_u and T_v perform an alignment procedure to compute a matrix that approximates T_v using T_u and vice versa. We build our graph with the arcs (u, v) and (v, u) that correspond to successful registrations.

3. Apply a greedy algorithm to select base meshes and insert new arcs into our graph. It stops when we find a dominating set.

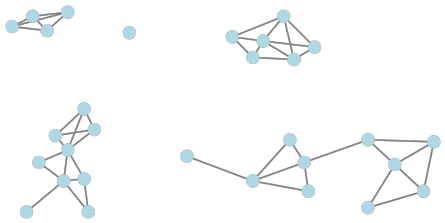
In Fig. 4.1, we can see an example of how the algorithm processes the feature vectors taken from the autoencoder. The first stage produces pairs of potential arcs from the feature vectors of the geometries. The second stage performs the registrations for each pair of neighbors and selects the transformations that respect the error thresholds to build our directed graph. Finally, the third stage greedily inserts new arcs into the graph and stops when it has found a dominating set.



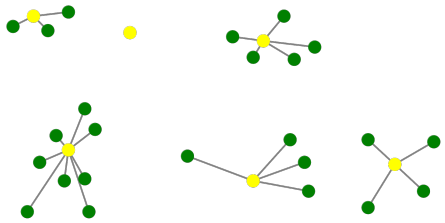
(a) Each dot represents a feature vector extracted from the autoencoder when fed with the point cloud sampled on the surface of a mesh.



(b) In the first stage, we find the K nearest neighbors for each mesh.



(c) In the second stage, we compute the registrations and select the successful ones to become arcs of a directed graph. This image shows an undirected graph for simplification purposes.



(d) The third stage generates the output of our algorithm by greedily selecting the set of base meshes. Selected base meshes are nodes in yellow, and their instances are nodes in green.

Figure 4.1: Simplified visualization of the stages of our algorithm.

Our models are composed of triangle meshes centered at the origin, then rotated using a Principal Component Analysis (PCA) technique, and finally scaled so that all their vertex coordinates range in the $[-0.5, 0.5]$ interval. These normalized geometries are densely sampled using the method proposed by Osada et al. (48). These are characteristics kept from the framework developed by Figueiredo et al. (5). However, our method changes the clustering and final registration steps. In Fig. 4.2, we display an overview of the adapted shape-matching framework we use to test our algorithm.

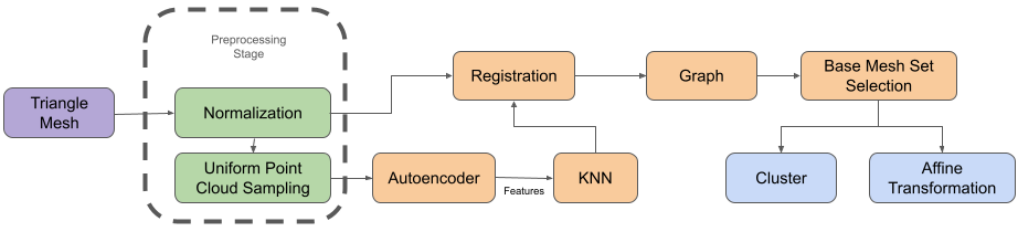


Figure 4.2: Unsupervised shape-matching framework based on the method proposed by Figueiredo et al. (5)

4.1.1 First stage

The dense point clouds sampled on the surfaces of our normalized meshes are given as input to a pre-trained autoencoder based on PointNet++ that extracts feature vectors for each point cloud. At this moment, each feature vector corresponds to a triangle mesh from the original model. In this latent space, we find for each geometry its K nearest neighbors. Where K is a hyperparameter to our algorithm. We compute the neighbors using the Euclidean distance in the feature space.

The output of the first stage of our algorithm is a set of pairs of geometries. Each pair indicates two geometries that were found to be sufficiently close neighbors according to the KNN algorithm.

4.1.2 Second stage

The second stage of our algorithm involves computing a matrix that transforms a base mesh into a candidate instance. This requires the usage of a registration procedure and the definition of error functions. The registration procedure proposed by Figueiredo et al. (9) takes two normalized meshes as input: a base mesh T_B and a target mesh T. We employ this procedure to find a matrix M that approximates the normalized target mesh with the normalized base mesh, such that the surface error function E_S (Equation 3-2) between a dense point cloud P_{MT_B} sampled on the surface of $M * T_B$ to the target mesh T is minimized.

A registration of $M * T_B$ into T will be rejected if the final error is above given thresholds. The functions evaluated are the symmetric surface error $SymE_S$ (Equation 3-3) and the maximum distance error $SymE_{max}$ (Equation 3-4). Additionally, the final area cannot differ more than a threshold given in squared meters.

$$E_{max}(Q,T) = \max_{q \in Q}(D_{\min}(q,T)) \tag{4-1}$$

At the end of each registration procedure, we consider T an instance of T_B when $SymE_S(M * T_B, T) \leq 0.05$, $SymE_{max}(M * T_B, T) \leq 0.07$ and the absolute total area difference is below 0.01 squared meters. These values are the same fine-tuned parameters reported by Figueiredo et al. (5).

We highlight that even though the error functions are symmetric, they are evaluated in the local space of the target mesh. The matrix M represents an affine transformation that does not preserve distances and, therefore, cannot guarantee that the error thresholds between $M^{-1} * T$ and T_B are met. For

that reason, we perform the registration procedure between neighbors in both directions.

4.1.3 Third stage

Starting from the graph produced in the second stage, the third stage of the algorithm extracts a set of clusters, each consisting of a base mesh and its instances. For that, we consider the strongly connected components individually. We propose the greedy algorithm presented in Algorithm 1 to select a set of base meshes for each strongly connected component. This algorithm initializes an empty set of base meshes and uses a greedy criterion to update it. In each iteration, new arcs starting in the selected base mesh are tested and then added to our graph. This algorithm stops when it has found a dominating set for the current connected component. Instead of computing new matrices using the previously presented alignment procedure, we traverse the graph to extend our knowledge of instances using the method detailed in Algorithm 2. This way, we avoid performing the costly operation of computing transformation matrices again.

Algorithm 1: Greedy Base Mesh Selection

Input:

G: Graph
nodes: nodes of the current connected component

Output:

B: Base mesh set picked among *nodes*

```
1  C ← Subgraph(G, nodes)
2  B ← ∅
3  while ¬IsDominatingSet(B, C) do
4      newBase ← PickBaseMesh(G, nodeSubset)
5      B ← B ∪ {newBase}
6      for each node t of G do
7          M ← ComposeMatrix(G, newBase, t)
8          if ErrorIsAcceptable(M, newBase, t) then
9              AddArc(G, < newBase, t >)
10         coveredSet ← Neighbors(G, B) ∪ B
11         nodeSubset ← nodeSubset \ coveredSet
```

We aim to insert as few new arcs as necessary and stop as soon as we find a dominating set for each connected component. If only one base mesh can represent a connected component, our picking criterion may pick this base mesh and stop after the first iteration. Our algorithm ends when all connected

components have been processed, and there is a dominating set for the whole graph.

Algorithm 2: Matrix Composition

Input:

G: Connected component
s: Node representing a candidate base mesh
t: Node representing a possible instance

Output:

M: Matrix that approximates t with s

Initialize the matrix M with an identity matrix
Find shortest p path from node s to node t
for *arc a in path p* **do**
    Update matrix M with matrix associated to a

4.1.3.1 Virtual arcs

Arcs represent successful registrations, so we assume that the existence of two arcs (u, w) and (w, v), with associated matrices M_{uw} and M_{wv}, respectively, suggests the possibility of adding a virtual arc (u, v) with matrix $M_{uv} = M_{wv} * M_{uw}$. We aim to insert virtual arcs into the connected components to find new pairs of valid registrations that can minimize the number of base meshes and further reduce the model's size. The graph may have multiple paths from a candidate base mesh to candidate instances. We can see this illustrated in Fig. 4.3. From u to v, there are multiple paths, and each path may result in a distinct transformation matrix. We found empirically that using an error function as arc weight, we can compute the shortest path to find a good enough registration matrix. Interestingly, the shortest path between two nodes sometimes enhanced the quality of successful registrations between initial neighbors. This means that we have improved some local minima of the optimization method.

The method for collecting the transformation matrix from a node s to a node t is presented in Algorithm 2. Virtual arcs are only added to the graph if their transformation respects the thresholds described by all previously mentioned error functions.

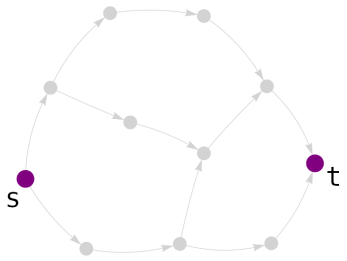


Figure 4.3: Illustration of possible paths between two different nodes

4.1.3.2 Greedy base mesh selection

We tested two picking criteria: the maximum degree and the shortest path. The first one is always to pick the node with the highest degree among the remaining nodes as defined in Algorithm 3. The second one consists of running the shortest path algorithm and adding up, for each node, the lengths of the shortest paths from it to all other nodes in the connected component so the node with the minimum sum is chosen. This is presented in Algorithm 4.

We must emphasize that we design these simple heuristics based on connectivity and arc weight because we strive to add as few arcs as necessary. The intuition behind the maximum degree is that a highly connected node may be more likely to have more instances in the connected component. The intuition behind the shortest path is that the sum of arc weight can be viewed as a rough estimation of the error between nodes farther apart, and the node with the minimum expected error may be a more promising choice, even though the actual error is not the sum of the arc errors.

Algorithm 3: Base mesh selection based on maximum degree

Input:

C: Connected component
nodes: Nodes of the current connected component

Output:

root: Node

```
1 rootDegree ← 0
2 for node v in nodes do
3     if degree(C, v) > rootDegree then
4         root ← v
5         rootDegree ← degree(C, v)
```

Algorithm 4: Base mesh selection based on shortest path

Input:

C: Connected component
$nodes$: Nodes of the current connected component

Output:

$root$: Node

```
1 rootDegree ← ShortestLengths(C)
2 minTotalLength ← ∞
3 for each node s of S do
4 |   acc ← 0
5 |   for each node t of S do
6 |   |   acc ← acc + lengths[s][t]
7 |   if acc < minTotalLength then
8 |   |   root ← s
```

5
Results

In this Chapter, we showcase the results our clustering algorithm achieved and compare them to the results of previous approaches. We describe the data sets we used and detail how to train an autoencoder. We display the values we used as parameters for our algorithm and their impact on our results. We show that our results had a significant improvement over previous approaches even with a small number of neighbors. We compare our algorithm to a fully supervised approach by applying it to a labeled data set and find that we significantly bridge the gap between the unsupervised and supervised approaches. We present an upper bound to our greedy base-mesh picking algorithm. Lastly, we discuss our method and its advantages to prior works.

The experiments were conducted with packages commonly used in Python, such as PyTorch, NetworkX, Numpy, and Scikit-Learn.

5.1
Data set description

Two data sets were used: a labeled data set composed of 18,851 geometries classified with 16 different labels and another data set comprising an amount of 61,561 unlabeled geometries of 5 different 3D CAD models. The unlabeled data set has geometries with diverse shapes and sizes. Similar geometries might have been modeled by different professionals using various tools, which makes the data set representative of the models we aim to reduce. These data sets were the same used by Figueiredo et al. (5). The labeled data set comprises two splits, one for training and one for testing. The unlabeled data set is composed of two halves organized in three splits. The first half is divided into a train set and a validation set. The second half is the input to our shape-matching algorithm. In this work, we also produced model reductions on the same splits used by Figueiredo et al. (5). In order to execute our algorithm, we used the PointNet++ autoencoder trained on the train split of the unlabeled dataset.

5.2
Autoencoder training

The first halves of the unlabeled data set were used for training and validation. Each first half was split into train and validation sets following a 90%/10% respective proportion. The autoencoder was trained with the combined train sets of the five models for 100 epochs, using the Adam optimizer

and a value of 0.001 for the learning rate. This training is the same as was described by Figueiredo et al. (5).

Then, the autoencoder was selected using the best mean reconstruction quality in the validation set among different epochs. The mean reconstruction quality on both train and validation sets was $1e^{-5}$, showing that the autoencoder was able to learn relevant features of the input data.

5.3 Experiments

Among the three error functions proposed by Figueiredo et al. (5) we found that the maximum distance error $SymE_{max}$ was the most likely to discard a registration. Even though its threshold was higher than the symmetric average distance error $SymE_S$, 0.07 as opposed to 0.05, it was still a more restrictive error evaluation. Therefore, we used this error value as arc weight in our graphs throughout this work.

We tested our method on the second halves of the five models of our unlabeled data set. We chose values for K as 3 and 9. As expected, a greater value for K improved model reduction. By comparing each geometry to a broader neighborhood, our algorithm found more similarities among geometries and as a result, fewer clusters. This can be seen in Table 5.1.

Our initial concern was that small values of K might result in too few comparisons. However, to our surprise, a value of $K = 3$ already performed surprisingly well. Compared to previous approaches on the same data (5), we improved the best average result by almost 9 percent points. The larger neighborhood of $K = 9$ performed even better, pushing our results further to 10 percent points over the best average of previous approaches.

Table 5.1: Comparative of Model Reductions Using Different Techniques.

	Max Degree		Shortest Path			
	K=9	K=3	K=9	K=3	Figueiredo et al. (5)	Ivson et Celes (8)
Model 1	**87.94%**	86.35%	87.36%	86.00%	76.58%	51.89%
Model 2	84.48%	83.31%	**84.55%**	83.30%	78.32%	65.66%
Model 3	**89.98%**	88.91%	89.62%	88.60%	80.71%	72.29%
Model 4	**88.22%**	87.21%	88.03%	87.11%	70.31%	82.15%
Model 5	**88.48%**	87.22%	88.30%	87.27%	82.25%	88.09%
Average	**87.82%**	86.60%	87.57%	86.45%	77.63%	72.01%

We remark that even accounting for individual models, no other unsupervised method performed better than ours. On Model 5, our method was marginally better than the method by Ivson et Celes (8), the second-best performing method on this model. From Fig. 5.1 to 5.5 the second halves of each model of the unlabeled dataset are displayed. We present three images for

each model: the original model, the colored model with instances detected via K-Means‖, and the instances detected with our graph-based approach.

To analyze the existing gap between the results of supervised shape matching (9) and our unsupervised shape-matching technique we performed our method in the aforementioned labeled data set. This data set has regular geometries of 16 classes. Figueiredo et al. (5) reports a 97.03% reduction on its test set with a supervised classification algorithm; 97.14% reduction when employing Ivson and Celes (8) method and an 87.07% reduction when using their unsupervised shape matching algorithm with a PointNet++ autoencoder trained on the unlabeled data set. On the other hand, our method with a neighborhood parameter K set to 9 and Maximum Degree as the base mesh picking criterion demonstrated significant improvement, achieving a 95.08% reduction using the latent space defined by the same autoencoder. This means a substantial advancement in bridging the gap between unsupervised and supervised shape-matching methods. Figure 5.6 illustrates the parametric model and the visualization of its identified instances.

Luckily, we have a simple manner of evaluating our greedy choice of base meshes. In the best scenario, the connected components generated in the second stage of our method could be represented by the smallest mesh. Let the lower bound on the model size be L. We can compute it with the sum of the best possible sizes for base meshes and the total size accounting for the 4x4 matrices required to represent every geometry in the model.

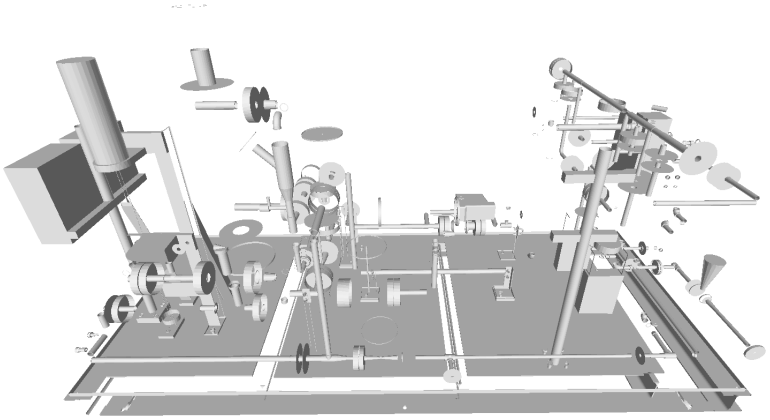
Below, s is a function that denotes the size of a mesh represented by node u, $s_{\text{4x4 matrix}}$ accounts for the size of an individual 4x4 matrix, $C(G)$ is the set of connected components of G and V is the set of vertices of a graph.

$$L = \left(\sum_{C_i \in C(G)} \underset{u \in V(C_i)}{min} \left(s\left(u\right) \right) \right) + |G| * s_{\text{4x4 matrix}} \tag{5-1}$$

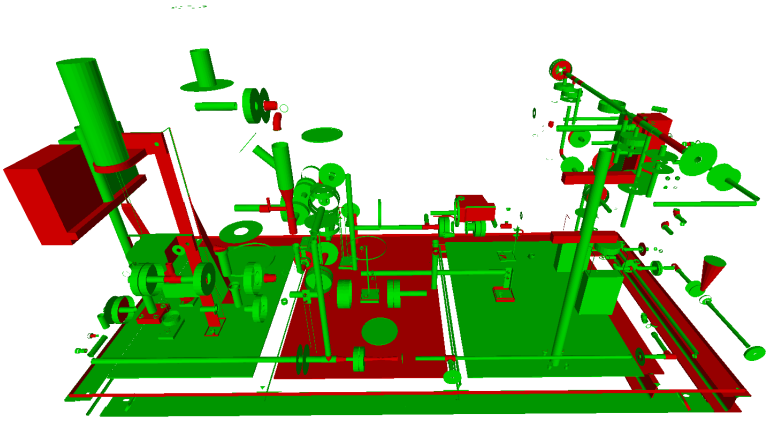
The expression below gives an upper bound R on the reduction, where we compute what the reduction would be in case we achieve the lower bound on model size.

$$R = 1 - \frac{L}{\left(\sum_{u \in V(G)} s(u) \right) + |G| * s_{\text{4x4 matrix}}} \tag{5-2}$$

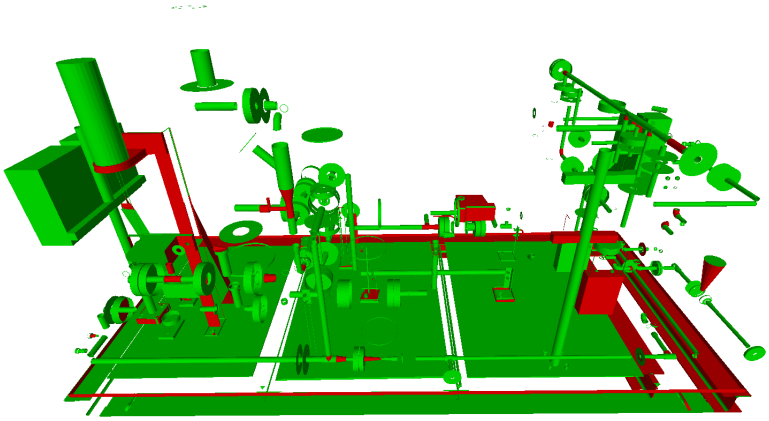
In Table 5.2, we can see that an optimal solution to the Minimum Dominating Set cannot increase the model reduction of the connected components generated by a $K = 9$ neighborhood even 2 percent points in Model 3. The distance from the upper bound to the results we achieved in the remaining models is even narrower.



(a) Original Model 1

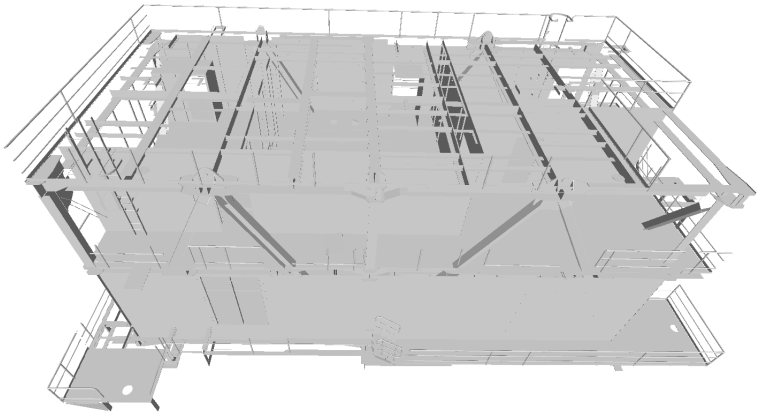


(b) Instance detection with KMeans‖ (5)

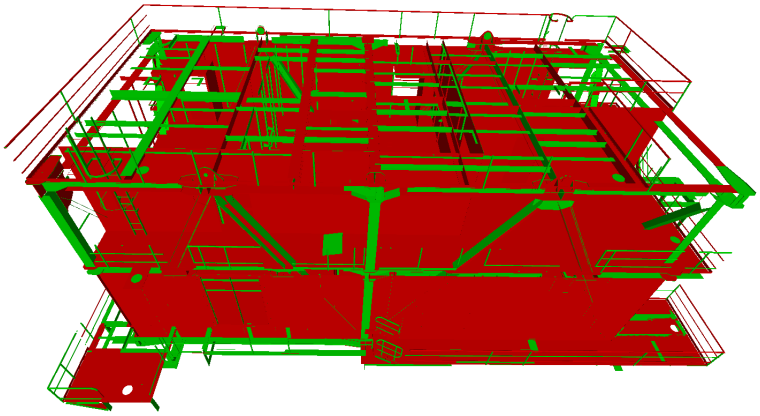


(c) Instance detection with our graph-based method

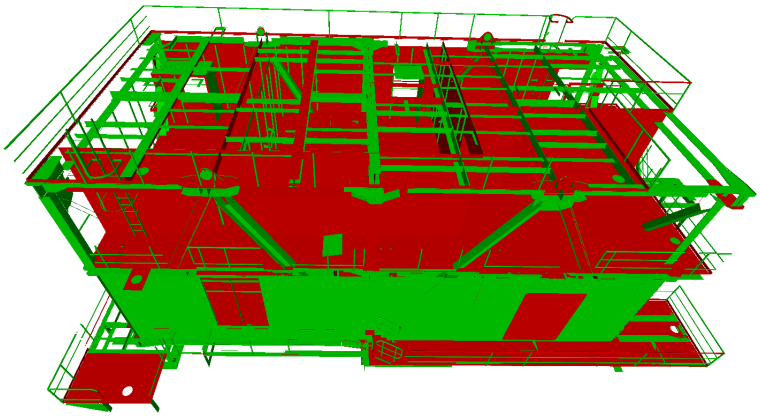
Figure 5.1: Second half of Model 1. Green geometries represent instance meshes, and red geometries are base meshes.



(a) Original Model 2

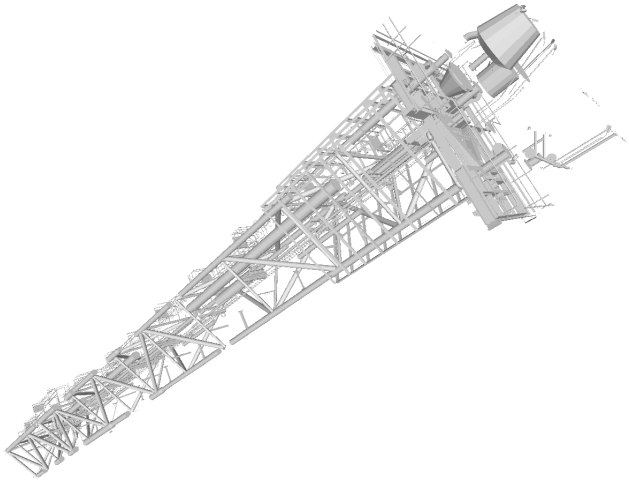


(b) Instance detection with KMeans‖ (5)

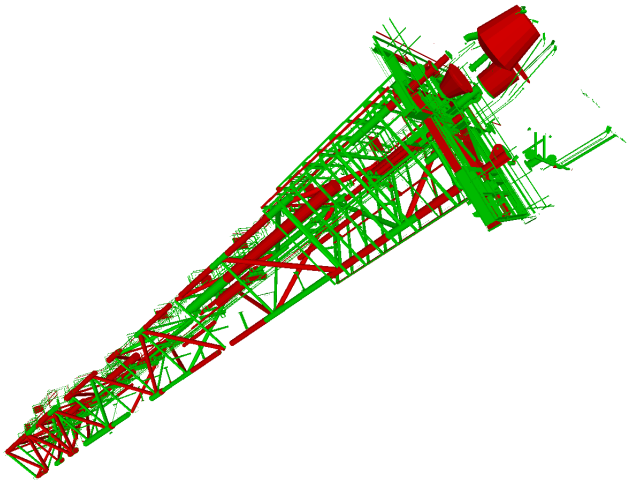


(c) Instance detection with our graph-based method

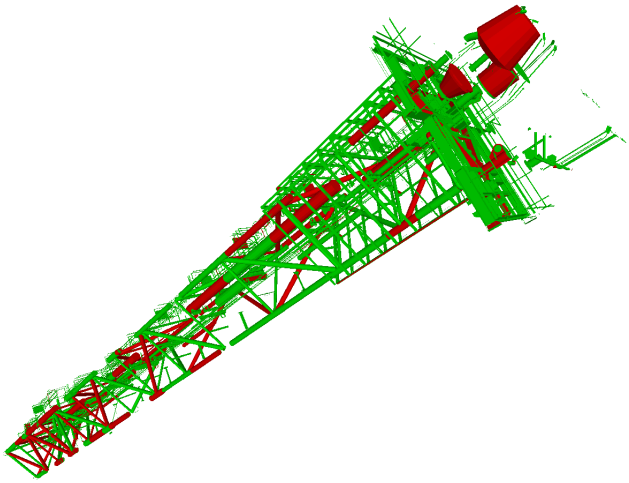
Figure 5.2: Second half of Model 2. Green geometries represent instance meshes, and red geometries are base meshes.



(a) Original Model 3

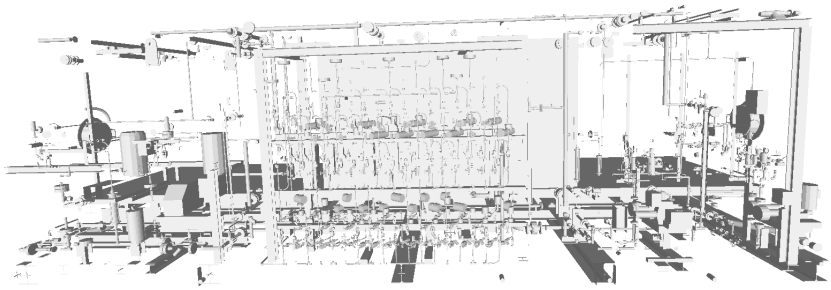


(b) Instance detection with KMeans‖ (5)

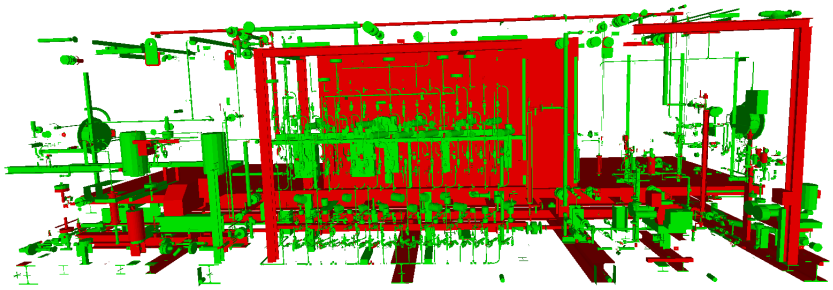


(c) Instance detection with our graph-based method

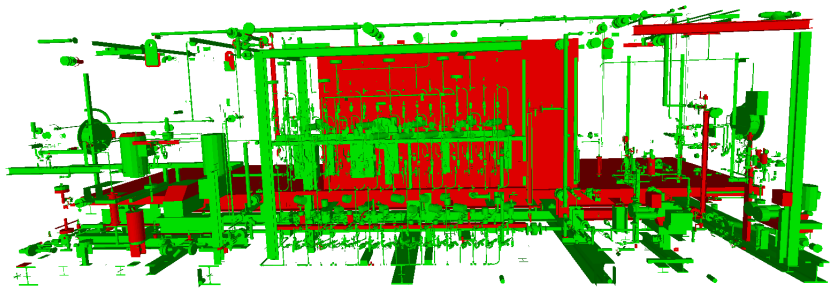
Figure 5.3: Second half of Model 3. Green geometries represent instance meshes, and red geometries are base meshes.



(a) Original Model 4

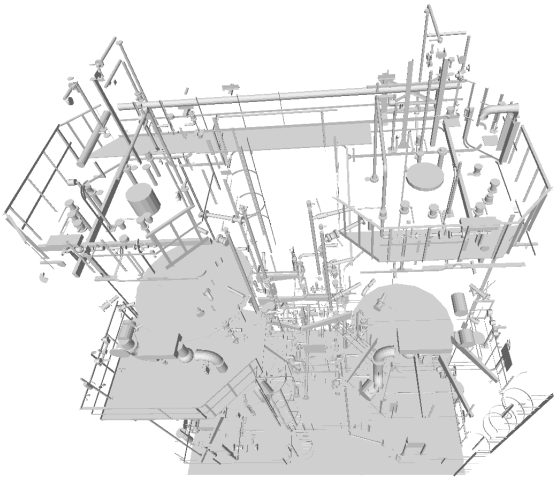


(b) Instance detection with KMeans‖ (5)

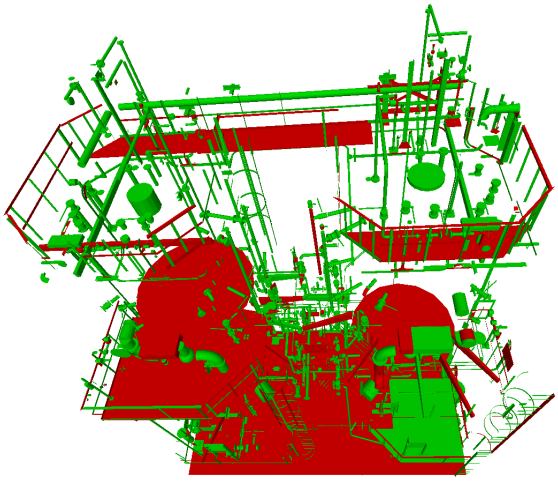


(c) Instance detection with our graph-based method

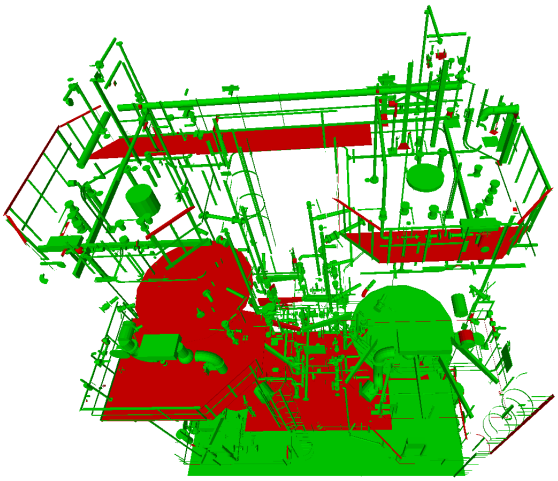
Figure 5.4: Second half of Model 4. Green geometries represent instance meshes, and red geometries are base meshes.



(a) Original Model 5



(b) Instance detection with KMeans‖ (5)



(c) Instance detection with our graph-based method

Figure 5.5: Second half of Model 5. Green geometries represent instance meshes, and red geometries are base meshes.

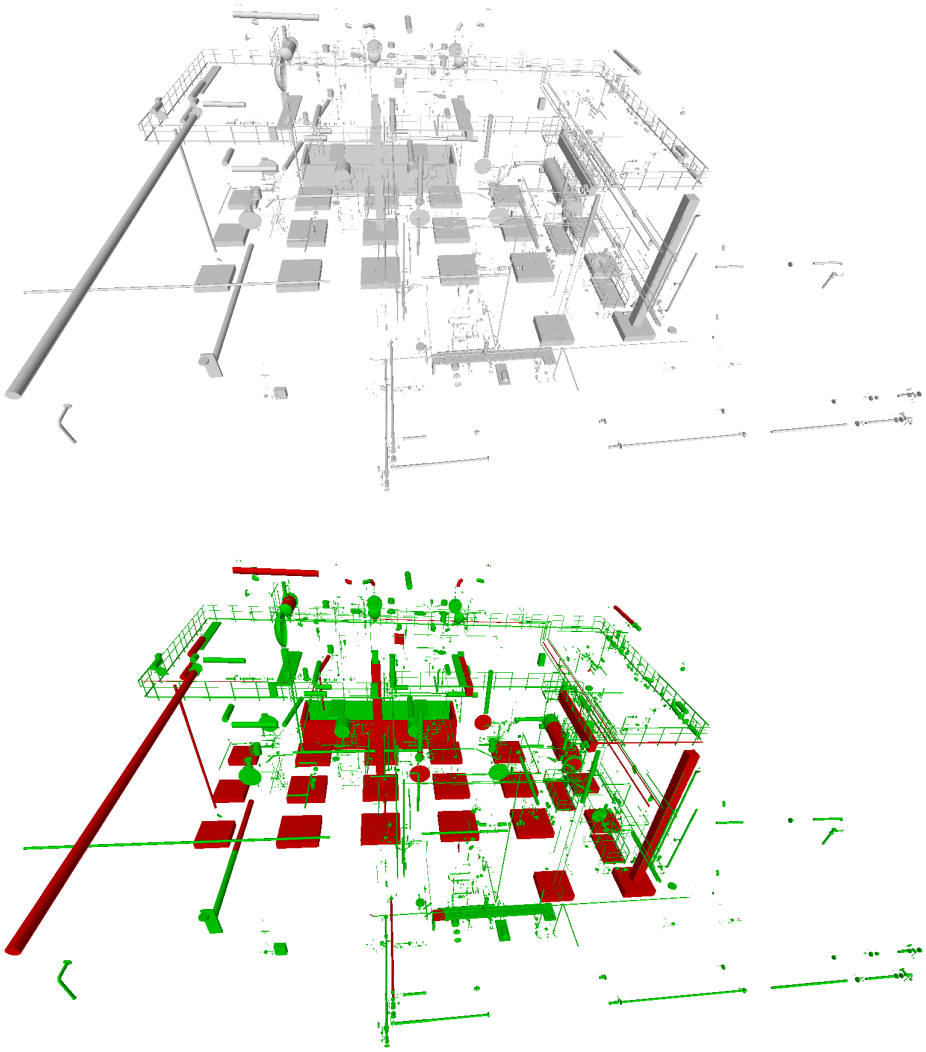


Figure 5.6: Test split of the labeled data set. The original model is above, and the model with instance information is below. Geometries in red are base meshes and geometries in green are accepted instances.

Table 5.2: A comparison between actual results from our greedy base mesh picking algorithms to the upper bound of model reductions for connected components generated by a $K = 9$ neighborhood.

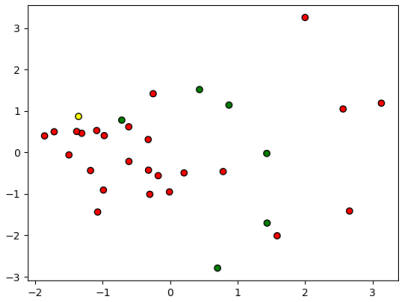
	Max Degree	Shortest Path	Upper bound
Model 1	87.94%	87.36%	88.87%
Model 2	84.48%	84.55%	85.40%
Model 3	89.98%	89.62%	91.74%
Model 4	88.22%	88.03%	88.72%
Model 5	88.48%	88.30%	89.34%

5.4 Discussion

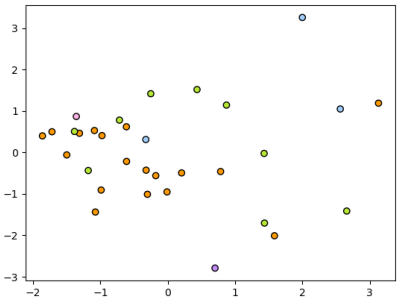
The most notable advantage of our algorithm is that it assigns nodes to clusters only in the last stage when we have already collected subsets of elements that are close in the feature space and share similar geometries. On the other hand, the original framework developed by Figueiredo et al. (5) has an unsupervised clustering step that only considers the points representing meshes in feature space. Their method relies on characteristics of the clustering algorithms to pick reference meshes. For instance, for K-Means‖, their reference mesh is the one closest to the centroid of each cluster. This leads to some clusters with very low instance ratios because meshes in the same cluster are either instances of the selected reference mesh or unique meshes. In addition, performing the registration as an intermediate step enables us to improve our choice of base meshes using the error and actual instance information, which is not the case in the clustering step of previous work.

In Fig. 5.7, we can see feature vectors projected onto two dimensions using PCA. Initially, these vectors were grouped into the same cluster by a K-Means‖ algorithm. However, this cluster had a poor conversion ratio because many geometries did not match the selected reference mesh. In contrast, our method could identify more instances and assign them to different clusters accordingly.

These features were derived from Model 2, and each clustering considered all geometries within this model. Hence, the colored dots in 5.7b do not necessarily represent entire clusters. It's evident that many instances disregarded by the previous method were recognized as instances of common base meshes. There is a clear contrast between both results, as our algorithm placed the same meshes into different clusters because many were instances of neighboring geometries. It's worth noting that clusters with low conversion are also present when using HDBSCAN (25).



(a) The nodes in red represent discarded instances of the chosen base mesh, represented in yellow. Green dots represent successfully identified instances of the base mesh.



(b) The same meshes of Fig. 5.7a, assigned to different clusters. Each color represents a different cluster generated by our algorithm, with a parameter of K=3.

Figure 5.7: A comparison between different cluster assignments of feature vectors originally placed in the same cluster by a K-Means‖ in 5.7a, and cluster assignments performed by our algorithm in 5.7b.

In our case, intuitively, each connected component would directly correspond to a cluster since the arcs represent successful registrations between pairs of nodes. However, only one reference mesh is often insufficient to represent a whole connected component within the error tolerance because the error usually increases between nodes farther apart. That is why we use the greedy base mesh selection in Algorithm 1.

For comparison, we calculated the ratio of clusters identified per number of geometries. We found that in all models, the number of clusters significantly differed depending on both K and the model. In Table 5.3, we can see how a neighborhood of $K = 3$ can significantly reduce the total meshes required to represent the model information. In Table 5.4, the number of total meshes is reduced even further. In these tables, singular meshes account for the meshes that had no successful registration in the first stage of our algorithm, resulting in connected components of a single node in the second stage. In contrast, total meshes are the number of clusters generated in the third stage and include singular meshes.

Given that the CAD domain typically consists of numerous repeated instances of the same base geometries, it is unsurprising that models with more geometries, such as Model 3 and Model 4 (each exceeding 10,000), exhibit a lower total mesh ratio. Also, the difference between singular meshes from a neighborhood of $K = 3$ to one of $K = 9$ is worth of note. This decrease indicates the importance of comparing a mesh to a significant number of neighbors to identify instances correctly. This need may arise due to utilizing

Table 5.3: Number of base meshes for each model found by our algorithm when $K = 3$ and Maximum Degree

	Singular meshes	Total meshes	Geometries	Ratio
Model 1	62	136	1581	8.6%
Model 2	77	193	2598	7.4%
Model 3	97	371	10332	3.6%
Model 4	122	423	11323	3.7%
Model 5	119	338	4947	6.8%

Table 5.4: Number of base meshes for each model found by our algorithm when $K = 9$ and Maximum Degree

	Singular meshes	Total meshes	Geometries	Ratio
Model 1	58	122	1581	7.7%
Model 2	64	163	2598	6.3%
Model 3	91	288	10332	2.8%
Model 4	89	318	11323	2.8%
Model 5	88	266	4947	5.4%

feature vectors from normalized geometries because the normalization process leads to the loss of finer details.

Even though our approach has several important properties, it still has some limitations. For instance, in the first stage of our algorithm, our choice for K does not guarantee that there will be a comparison between every possible instance for a base mesh. Sub-optimal clusters will be generated if instances are not in the same initial connected components. A result that hints this happened in our experiments is the upper bound of reduction on the labeled data set, which was 95.79%. By comparison, we know that it is possible to reduce the size of this model by up to 97.14%, a result that Ivson and Celes' method (8) achieved.

Another factor that must be taken into account is the execution time of our algorithm. The first-stage execution time is negligible. Hence, we analyze the second and third stages of our algorithm. In Table 5.5 we see the total execution time of the second stage for each model for different values of K, the only variable that affects the execution time of this stage. We observe that a larger neighborhood resulted in an almost proportional increase in the execution time. However, this increase does not result in a proportional efficiency to our algorithm, as we have seen in Table 5.1. Therefore, a smaller value for K is advantageous in the sense that it produces good enough results in less time.

In Table 5.6, we see the elapsed execution time for our two greedy strategies and their different values for K. We notice that a bigger neighborhood

Table 5.5: Second-stage total execution time for each model (hours)

	K=3	K=9
Model 1	33.1	95.1
Model 2	57.3	165.4
Model 3	213.6	614.2
Model 4	233.0	668.7
Model 5	110.3	319.6

resulted in a graph with more initial edges, which results in an increase in elapsed time due to the shortest path algorithm used to compute the virtual edges. When comparing the elapsed time for graphs with the same initial number of edges we see that the shortest path took more time to execute because it is more complex than simply picking the node with the maximum degree, but it can be faster than the maximum degree strategy when it finds a better set of base meshes, for example, in Model 2 with $K = 9$.

Table 5.6: Third-stage total elapsed time for each model (hours)

	Max Degree		Shortest Path	
	K=3	K=9	K=3	K=9
Model 1	1.0	1.1	1.0	1.2
Model 2	1.7	2.2	1.9	2.0
Model 3	14.8	17.1	17.6	18.7
Model 4	6.7	9.0	6.8	9.6
Model 5	1.6	1.8	1.8	2.1

6
Conclusion and Future Work

In this work, we presented a three-stage graph-based clustering algorithm that improved unsupervised clustering in deep feature space and applied it to a shape-matching framework (5). Also, as we build on top of existing work, we retain its advantages, such as preserving the geometries' intrinsic characteristics, being invariant to vertex ordering, and avoiding manual annotation on large data sets.

Our work resulted in a further reduction of memory requirements for the 3D models. It achieved a 95% model reduction in a labeled dataset, outperforming previous unsupervised techniques that reached 87% and getting almost as far as the 97% reduction from a fully supervised approach, effectively bridging the gap between both methods. In an unlabeled dataset, our method attains an average model reduction of 87% versus an average reduction of 77% from previous unsupervised techniques.

For future work, we observe that the first stage of our algorithm could be improved if we combine our approach with other unsupervised clustering approaches, such as algorithms that build hierarchies like HDBSCAN (25), Girvan-Newman (49) or Louvain method (50). This way, we could further ensure the quality of the connected components processed in the following stages of our algorithm and avoid sub-optimal upper bounds on the reduction of models.

Another course of action would be to test a metric as an error function because metrics have guaranteed triangular inequality, so we may save computations when adding virtual edges to our graph. The Hausdorff distance is an example of a metric that could be used for this purpose.

Since we rely on a PointNet++ (3) autoencoder, future work could test other architectures for unsupervised feature learning on point clouds (51, 52, 53) as well as other loss functions (54) to improve the learned latent space and consequently shape matching accuracy.

7
Bibliography

1 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

2 CHARLES, R. Q. et al. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Honolulu, HI: IEEE, 2017. p. 77–85. ISBN 978-1-5386-0457-1. Disponível em: <http://ieeexplore.ieee.org/document/8099499/>.

3 QI, C. R. et al. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. **Advances in neural information processing systems**, v. 30, p. 10, 2017.

4 RAYKOV, Y. P. et al. What to Do When K-Means Clustering Fails: A Simple yet Principled Alternative Algorithm. **PLOS ONE**, v. 11, n. 9, p. e0162259, set. 2016. ISSN 1932-6203. Disponível em: <https://dx.plos.org/10.1371/journal.pone.0162259>.

5 FIGUEIREDO, L.; IVSON, P.; CELES, W. Unsupervised method for identifying shape instances on 3D CAD models. **Computers & Graphics**, v. 116, p. 228–238, nov. 2023. ISSN 00978493. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0097849323001875>.

6 AZHAR, S.; KHALFAN, M.; MAQSOOD, T. Building information modeling (bim): now and beyond. **The Australasian Journal of Construction Economics and Building**, UTS ePress, Broadway, NSW, Australia, v. 12, n. 4, p. [15]–28, 2012. Disponível em: <https://search.informit.org/doi/10.3316/informit.013120167780649>.

7 JIANG, Y. et al. Industrial applications of digital twins. **Philosophical Transactions of the Royal Society A**, The Royal Society Publishing, v. 379, n. 2207, p. 20200360, 2021.

8 IVSON, P.; CELES, W. Instanced Rendering of Massive CAD Models Using Shape Matching. In: **2014 27th SIBGRAPI Conference on Graphics, Patterns and Images**. Rio de Janeiro: IEEE, 2014. p. 335–342. ISBN 978-1-4799-4258-9. Disponível em: <https://ieeexplore.ieee.org/document/6915326/>.

9 FIGUEIREDO, L.; IVSON, P.; CELES, W. Deep learning-based framework for Shape Instance Registration on 3D CAD models. **Computers & Graphics**, v. 101, p. 72–81, dez. 2021. ISSN 00978493. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S009784932100176X>.

10 GAREY, M. R.; JOHNSON, D. S. Computers and intractability. **A Guide to the**, 1979.

11 DESAI, M.; SHAH, M. An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (MLP) and Convolutional neural network (CNN). **Clinical eHealth**, v. 4, p. 1–11, 2021. ISSN 2588-9141. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2588914120300125>.

12 HEIDARI, A. A. et al. An efficient hybrid multilayer perceptron neural network with grasshopper optimization. **Soft Computing**, v. 23, n. 17, p. 7941–7958, set. 2019. ISSN 1433-7479. Disponível em: <https://doi.org/10.1007/s00500-018-3424-2>.

13 CYBENKO, G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, v. 2, n. 4, p. 303–314, dez. 1989. ISSN 1435-568X. Disponível em: <https://doi.org/10.1007/BF02551274>.

14 LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: TOURETZKY, D. (Ed.). **Advances in Neural Information Processing Systems**. Morgan-Kaufmann, 1989. v. 2. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf>.

15 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet classification with deep convolutional neural networks. **Communications of the ACM**, v. 60, n. 6, p. 84–90, maio 2017. ISSN 0001-0782, 1557-7317. Disponível em: <https://dl.acm.org/doi/10.1145/3065386>.

16 College of Electronic Science and Engineering, National University of Defense Technology et al. Convolutional neural networks for time series classification. **Journal of Systems Engineering and Electronics**, v. 28, n. 1, p. 162–169, fev. 2017. ISSN 10044132. Disponível em: <http://ieeexplore.ieee.org/document/7870510/>.

17 ZEILER, M. D.; FERGUS, R. **Visualizing and Understanding Convolutional Networks**. arXiv, 2013. ArXiv:1311.2901 [cs]. Disponível em: <http://arxiv.org/abs/1311.2901>.

18 XU, W. et al. Variational Autoencoder for Semi-Supervised Text Classification. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 31, n. 1, fev. 2017. ISSN 2374-3468, 2159-5399. Disponível em: <https://ojs.aaai.org/index.php/AAAI/article/view/10966>.

19 ZHANG, Z.; SONG, Y.; QI, H. **Age Progression/Regression by Conditional Adversarial Autoencoder**. arXiv, 2017. ArXiv:1702.08423 [cs]. Disponível em: <http://arxiv.org/abs/1702.08423>.

20 YANG, X. et al. **Deep Spectral Clustering using Dual Autoencoder Network**. arXiv, 2019. ArXiv:1904.13113 [cs, stat]. Disponível em: <http://arxiv.org/abs/1904.13113>.

21 LI, C.-L. et al. **Point Cloud GAN**. arXiv, 2018. ArXiv:1810.05795 [cs, stat]. Disponível em: <http://arxiv.org/abs/1810.05795>.

22 XU, D.; TIAN, Y. A Comprehensive Survey of Clustering Algorithms. **Annals of Data Science**, v. 2, n. 2, p. 165–193, jun. 2015. ISSN 2198-5804, 2198-5812. Disponível em: <http://link.springer.com/10.1007/s40745-015-0040-1>.

23 BAHMANI, B. et al. **Scalable K-Means++**. 2012.

24 ARTHUR, D.; VASSILVITSKII, S. et al. k-means++: The advantages of careful seeding. In: **Soda**. [S.l.: s.n.], 2007. v. 7, p. 1027–1035.

25 CAMPELLO, R. J. G. B.; MOULAVI, D.; SANDER, J. Density-based clustering based on hierarchical density estimates. In: PEI, J. et al. (Ed.). **Advances in Knowledge Discovery and Data Mining**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 160–172. ISBN 978-3-642-37456-2.

26 ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: **kdd**. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231.

27 YOUNG, N. E. Greedy set-cover algorithms (1974-1979, chvátal, johnson, lovász, stein). **Encyclopedia of algorithms**, Springer Heidelberg, p. 379–381, 2008.

28 HASSIN, R.; LEVIN, A. A Better-Than-Greedy Approximation Algorithm for the Minimum Set Cover Problem. **SIAM Journal on Computing**, v. 35, n. 1, p. 189–200, jan. 2005. ISSN 0097-5397, 1095-7111. Disponível em: <http://epubs.siam.org/doi/10.1137/S0097539704444750>.

29 CYGAN, M.; KOWALIK, Å.; WYKURZ, M. Exponential-time approximation of weighted set cover. **Information Processing Letters**, v. 109, n. 16, p. 957–961, jul. 2009. ISSN 00200190. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0020019009001719>.

30 ALBUQUERQUE, M.; VIDAL, T. An Efficient Matheuristic for the Minimum-Weight Dominating Set Problem. **Applied Soft Computing**, v. 72, p. 527–538, nov. 2018. ISSN 15684946. ArXiv:1808.09809 [cs]. Disponível em: <http://arxiv.org/abs/1808.09809>.

31 HABIBULLA, Y.; ZHAO, J.-H.; ZHOU, H.-J. The Directed Dominating Set Problem: Generalized Leaf Removal and Belief Propagation. In: . [s.n.], 2015. v. 9130, p. 78–88. ArXiv:1505.03537 [cond-mat, physics:physics]. Disponível em: <http://arxiv.org/abs/1505.03537>.

32 NAKKALA, M. R.; SINGH, A.; ROSSI, A. Swarm intelligence, exact and matheuristic approaches for minimum weight directed dominating set problem. **Engineering Applications of Artificial Intelligence**, v. 109, p. 104647, mar. 2022. ISSN 09521976. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0952197621004528>.

33 BOUAMAMA, S.; BLUM, C. A hybrid algorithmic model for the minimum weight dominating set problem. **Simulation Modelling Practice and Theory**, v. 64, p. 57–68, maio 2016. ISSN 1569190X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1569190X15001574>.

34 LIN, G.; ZHU, W.; ALI, M. M. An Effective Hybrid Memetic Algorithm for the Minimum Weight Dominating Set Problem. **IEEE Transactions on Evolutionary Computation**, v. 20, n. 6, p. 892–907, dez. 2016. ISSN 1089-778X, 1089-778X, 1941-0026. Disponível em: <http://ieeexplore.ieee.org/document/7426831/>.

35 LIN, G.; GUAN, J. A Binary Particle Swarm Optimization for the Minimum Weight Dominating Set Problem. **Journal of Computer Science and Technology**, v. 33, n. 2, p. 305–322, mar. 2018. ISSN 1000-9000, 1860-4749. Disponível em: <http://link.springer.com/10.1007/s11390-017-1781-4>.

36 MITRA, N. J.; GUIBAS, L. J.; PAULY, M. Partial and Approximate Symmetry Detection for 3D Geometry. **ACM Transactions on Graphics (ToG)**, ACM New York, NY, USA, v. 25, n. 3, p. 560–568, 2006.

37 GAL, R.; COHEN-OR, D. Salient geometric features for partial shape matching and similarity. **ACM Transactions on Graphics**, v. 25, n. 1, p. 130–150, jan. 2006. ISSN 0730-0301, 1557-7368. Disponível em: <https://dl.acm.org/doi/10.1145/1122501.1122507>.

38 ALT, H. et al. Congruence, similarity, and symmetries of geometric objects. **Discrete & Computational Geometry**, Springer, v. 3, n. 3, p. 237–256, 1988.

39 MARTINET, A. et al. Accurate detection of symmetries in 3D shapes. **ACM Transactions on Graphics**, v. 25, n. 2, p. 439–464, abr. 2006. ISSN 0730-0301, 1557-7368. Disponível em: <https://dl.acm.org/doi/10.1145/1138450.1138462>.

40 PAULY, M. et al. Discovering structural regularity in 3D geometry. In: **ACM SIGGRAPH 2008 papers**. Los Angeles California: ACM, 2008. p. 1–11. ISBN 978-1-4503-0112-1. Disponível em: <https://dl.acm.org/doi/10.1145/1399504.1360642>.

41 HORN, B. K. P. Closed-form solution of absolute orientation using unit quaternions. **Journal of the Optical Society of America A**, v. 4, n. 4, p. 629, abr. 1987. ISSN 1084-7529, 1520-8532. Disponível em: <https://opg.optica.org/abstract.cfm?URI=josaa-4-4-629>.

42 KANATANI, K. Analysis of 3-D rotation fitting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 16, n. 5, p. 543–549, maio 1994. ISSN 01628828. Disponível em: <http://ieeexplore.ieee.org/document/291441/>.

43 EGGERT, D.; LORUSSO, A.; FISHER, R. Estimating 3-D rigid body transformations: a comparison of four major algorithms. **Machine Vision and Applications**, v. 9, n. 5-6, p. 272–290, mar. 1997. ISSN 0932-8092, 1432-1769. Disponível em: <http://link.springer.com/10.1007/s001380050048>.

44 LI, L. et al. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In: **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. Long Beach, CA, USA: IEEE, 2019. p. 2647–2655. ISBN 978-1-72813-293-8. Disponível em: <https://ieeexplore.ieee.org/document/8953681/>.

45 KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. arXiv, 2017. ArXiv:1412.6980 [cs]. Disponível em: <http://arxiv.org/abs/1412.6980>.

46 ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: an efficient data clustering method for very large databases. **ACM sigmod record**, ACM New York, NY, USA, v. 25, n. 2, p. 103–114, 1996.

47 DERPANIS, K. G. Mean shift clustering. **Lecture Notes**, v. 32, p. 1–4, 2005.

48 OSADA, R. et al. Shape distributions. In: . [s.n.], 2002. v. 21, n. 4, p. 807 âĂŞ 832. Cited by: 1540. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-33645783985&doi=10.1145%2f571647.571648&partnerID=40&md5=269ab0f4c96f23db485c88cd6dd058c0>.

49 GIRVAN, M.; NEWMAN, M. E. Community structure in social and biological networks. **Proceedings of the national academy of sciences**, National Acad Sciences, v. 99, n. 12, p. 7821–7826, 2002.

50 BLONDEL, V. D. et al. Fast unfolding of communities in large networks. **Journal of statistical mechanics: theory and experiment**, IOP Publishing, v. 2008, n. 10, p. P10008, 2008.

51 YANG, G. et al. PointFlow: 3D Point Cloud Generation With Continuous Normalizing Flows. In: **2019 IEEE/CVF International Conference on Computer Vision (ICCV)**. Seoul, Korea (South): IEEE, 2019. p. 4540–4549. ISBN 978-1-72814-803-8. Disponível em: <https://ieeexplore.ieee.org/document/9010395/>.

52 GAO, X.; HU, W.; QI, G.-J. GraphTER: Unsupervised Learning of Graph Transformation Equivariant Representations via Auto-Encoding Node-Wise Transformations. In: **2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. Seattle, WA, USA: IEEE, 2020. p. 7161–7170. ISBN 978-1-72817-168-5. Disponível em: <https://ieeexplore.ieee.org/document/9156576/>.

53 SUN, Y. et al. PointGrow: Autoregressively Learned Point Cloud Generation with Self-Attention. In: **2020 IEEE Winter Conference on Applications of Computer Vision (WACV)**. Snowmass Village, CO, USA: IEEE, 2020. p. 61–70. ISBN 978-1-72816-553-0. Disponível em: <https://ieeexplore.ieee.org/document/9093430/>.

54 NGUYEN, T. et al. Point-set Distances for Learning Representations of 3D Point Clouds. In: **2021 IEEE/CVF International Conference on Computer Vision (ICCV)**. Montreal, QC, Canada: IEEE, 2021. p. 10458–10467. ISBN 978-1-66542-812-5. Disponível em: <https://ieeexplore.ieee.org/document/9711238/>.